Teemu Mäkelä

Varastonhallintajärjestelmän laitekommunikoinnin toteutus TCP-pistokkeilla

Opinnäytetyö
Kevät 2020
SeAMK Tekniikka
Automaatiotekniikan tutkinto-ohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Sähköautomaatio

Tekijä: Teemu Mäkelä

Työn nimi: Varastonhallintajärjestelmän laitekommunikoinnin toteutus TCP-pistokkeilla

Ohjaaja: Mikko Ylihärsilä

Vuosi: 2020 Sivumäärä: 47

Tässä opinnäytetyössä tutkittiin Pesmel Oy:n WMS-varastonhallintajärjestelmän laitekommunikoinnin toteutusta TCP-pistokkeilla. Tavoitteena oli selvittää, olisiko tällä toteutuksella mahdollista parantaa järjestelmän suorituskykyä.

Työn teoriaosuuteen on kerätty tietoa TCP/IP-protokollapinon teknisistä ominaisuuksista. Työhön kuuluu myös testisovellus, jonka avulla testataan viestiliikennettä PC:n ja PLC:n välillä. Sovellus koostui PC:llä ja PLC:llä ajettavista ohjelmista, joista molemmista löytyi TCP-pistokkeita käyttävä palvelin- ja asiakastoteutus.

Tutkimuksessa selvisi, että yhden viestin lähettäminen ja kuittauksen vastaanottaminen testisovelluksella kestää noin 100 ms, eli lähetysnopeus on noin 50 ms per viesti. Koska tällä toteutuksella viestejä lähetetään yksi kerrallaan, toteutus ei tuo haluttua suorituskykyparannusta.

Avainsanat: TCP, IP, TCP-pistoke, Java, PLC, laitekommunikointi

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Engineering

Specialisation: Electrical Automation

Author: Teemu Mäkelä

Title of thesis: Device Communication for a Warehouse Management System Using TCP Sockets

Supervisor: Mikko Ylihärsilä

Year: 2020 Number of pages: 47

The thesis studied the possibility of implementing a new type of device communication system for Pesmel WMS warehouse management system by using TCP sockets. The aim was to improve the performance of the WMS.

The thesis studied information regarding the technical specification of the TCP/IP protocol stack. The thesis also introduced a test application which is used for testing the messaging between a PC and PLC. The application consists of programs for PC and PLC which both include a server and a client implementation using TCP sockets.

The conclusion of the study was that sending a message and receiving an acknowledgement using the test application takes approximately 100 ms, so the rate of transmission is 50 ms for each message. Because the messages are sent one at a time, this implementation does not improve the performance as desired.

Keywords: TCP, IP, TCP socket, Java, PLC, device communication

SISÄLTÖ

Kuvio- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

API	Application Programming Interface eli ohjelmointirajapinta, joka määrittää kuinka ohjelmistot keskustelevat.
Asiakas	(Eng. client) Osa asiakas-palvelin-arkkitehtuuria, asiakkaan tehtävänä on lähettää pyyntöjä palvelimelle.
IP	Internet Protocol on verkkokerroksen protokolla, joka huolehtii verkkopakettien toimittamisesta perille.
JVM	Java Virtual Machine on virtuaalikone, joka suorittaa Java-kielellä tehtyjen ohjelmien sille käännettyä tavukoodia.
Konstruktori	(Eng. constructor) Luokan erikoismetodi, joka luo olion.
Luokka	(Eng. class) Määrittelee olion rakenteen.
Metodi	(Eng. method) Olioon liittyvä toiminnallisuus.
Olio	(Eng. object) Luokan ilmentymä, joka sisältää joukon yhteenkuuluvaa tietoa ja toiminnallisuutta.
OPC UA	Open Platform Communications Unified Architecture, teollisuusautomaation tiedonsiirtostandardi.
Palvelin	(Eng. server) Osa asiakas-palvelin-arkkitehtuuria, palvelimen tehtävänä on vastata asiakkaan pyyntöihin.
Pistoke	(Eng. socket) Rajapinta tiedon lähettämiseen ja vastaanottamiseen päätepisteiden välillä joko verkossa tai prosessien välillä.
PLC	Programmable Logic Controller eli ohjelmoitava logiikka on pieni uudelleen ohjelmoitava tietokone, jolla voidaan ohjata siihen kytkettyjä laitteita.

Säie	(Eng. thread) Pienin itsenäisesti hallittu yksikkö, jota yksi tai useampi prosessori voi ajaa useasta paikasta samanaikaisesti.
TCP	Transmission Control Protocol, yhteydellinen tiedonsiirtoprotokolla.
WMS	Pesmel Oy:n automatisoitu varastonhallintajärjestelmä.

1 JOHDANTO

1.1 Työn tausta

Yrityksen WMS-varastonhallintajärjestelmän laitekommunikointi on nykyisin toteutettu OPC UA -yhteydellä. Tämän opinnäytetyön aiheena on selvittää, onnistuuko yrityksen laitekommunikoinnin toteutus TCP-pistokkeilla ja parantaako se järjestelmän suorituskykyä.

OPC UA -yhteydellä on aiemmin yritetty käyttää tilaajaa, joka päivystää luettavia data tageja. Kaikki viestit eivät kuitenkaan ole tulleet perille, joten tilaaja on korvattu data tagien syklisellä lukemisella. Tagien jatkuva lukeminen kuitenkin laskee järjestelmän suorituskykyä. Tältä voitaisiin kuitenkin välttyä käyttämällä TCP-pistokkeita.

1.2 Työn tavoite

Tavoitteena on tehdä TCP-pistokkeilla toteutettu testiympäristö WMS-järjestelmän ja useiden ohjelmoitavien logiikoiden väliseen kommunikointiin. Ympäristöllä testataan datan lukeminen ja kirjoittaminen yhteyden molemmissa päissä. Lisäksi on huolehdittava siitä, että kaikki data menee perille. Ohjelman tulisi siis toteuttaa viestin uudelleenlähetys, jos vastapuoli ei kuittaa viestiä tai yhteys katkeaa.

Rakennetun testiympäristön avulla voidaan arvioida, että kannattaako TCP-pistokeyhteys ottaa käyttöön uusissa projekteissa.

1.3 Työn rakenne

Luku kaksi on työn teoriaosuus, jossa käsitellään TCP/IP-protokollapinoa.

Luvussa kolme käydään läpi työssä käytetyt laitteet, ohjelmistot ja ohjelmointikielet.

Luvussa neljä käsitellään työn testisovelluksen suunnittelua. Luvussa kerrotaan, mitä ominaisuuksia sovellukselta vaaditaan ja kuvaillaan sovelluksen toimintaa.

Luvuissa viisi ja kuusi esitellään testisovelluksen WMS- ja PLC-ohjelmien käytännön toteutus.

Luvussa seitsemän käydään läpi työn tulokset.

Luku kahdeksan on työn yhteenveto, jossa selvitetään työn vaiheet, pohditaan työn tuloksia sekä mietitään jatkokehitystä.

1.4 Pesmel Oy

Tutkimuksen toimeksiantaja on Pesmel Oy. Pesmel on suomalainen automatisoitujen materiaalinkäsittely-, lastaus- ja pakkausjärjestelmien toimittaja metalli-, paperi- ja rengasteollisuudelle. Pesmel toimittaa järjestelmiä maailmanlaajuisesti ja noin puolet toimituksista menee Aasian maihin. (Pesmel 2018.)

1.5 Pesmel WMS-varastonhallintajärjestelmä

WMS on Pesmel Oy:n automatisoitu varastonhallintajärjestelmä. Järjestelmä pitää yllä tietokantaa käsiteltävistä materiaaleista ja optimoi varastoon tulevat ja sieltä lähtevät materiaalivirrat. Järjestelmä muokataan asiakaskohtaisesti asiakkaan materiaaleihin, varastoihin, tuotantolinjoihin ja rajapintoihin sopivaksi.

2 TCP/IP-PROTOKOLLAPINO

2.1 Historia

Internet on toteutettu TCP/IP-protokollapinon avulla. Protokollat kehitettiin alun perin osana Yhdysvaltain puolustusministeriön DARPA-osaston (Defence Advanced Research Projects Agency) tutkimushanketta 1970-luvulla korvaamaan ARPANET-verkon Network Core Protocol -tekniikka, joka ei vastannut kasvavan verkon tarpeita. (Anttila 2001, 13-15.)

2.2 TCP/IP yleisesti

TCP/IP koostuu useista eri protokollista, joista tärkeimmät ovat TCP-protokolla ja IP-protokolla. TCP/IP-pino on nelikerroksinen, ja se vastaa osittain OSI-kerrosmallia. Sovelluskerros käsittää OSI-mallin sovellus-, esitystapa- ja yhteysjaksokerrokset. TCP-protokolla toteuttaa kuljetuskerroksen, ja IP-protokolla toteuttaa verkkokerroksen. Fyysinen kerros käsittää OSI-mallin kaksi alinta kerrosta. (Anttila 2001, 33-36.)

2.3 OSI-malli

OSI-malli on kansainvälisen standardointiorganisaation ISO:n 1980-luvulla kehittämä tiedonsiirtoprotokollien referenssimalli, jonka avulla voidaan mallintaa eri verkkoteknologioita. Malli kehitettiin ratkaisemaan eri verkkotekniikoiden yhteensopivuusongelmat sekä mahdollistamaan kaikkien maailman laitteiden yhdistämisen toisiinsa. (Anttila 2001, 33-36.)

OSI-kerrosmalli koostuu seitsemästä itsenäisestä kerroksesta, joilla kaikilla on oma tehtävänsä. Taulukossa 1 on esitelty kerrokset ja niiden tehtävät.

Taulukko 1. OSI-kerrosmalli (mukaillen Anttila 2001, 35.)

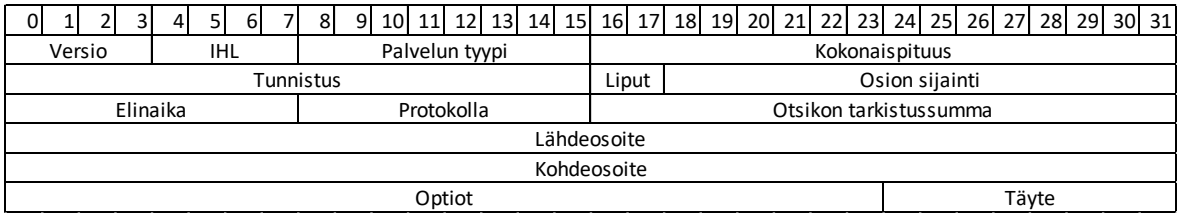
Kerroksen nimi	Kerroksen tehtävä
Sovelluskerros	Rajapinta, joka tarjoaa verkkopalveluja sovelluksille
Esitystapakerros	Määrittelee, millaisessa muodossa välitettävä data esitetään
Yhteysjaksokerros	Sovellusten toimintojen koordinointi laitteiden välillä
Kuljetuskerros	Vastaa ylemmiltä kerroksilta tulevan datan pilkkomisesta ja palasten välittämisestä vastaanottajalle
Verkkokerros	Pakkaa kuljetuskerrokselta saadun datan ja välittää sen vastaanottajalle
Siirtoyhteyskerros	Rakentaa kehyksen, johon pakataan verkkokerrokselta saatu data
Fyysinen kerros	Määrittelee verkkotekniikan fyysisen toteutuksen, esim. kaapelit ja liittimet

2.4 IP-protokolla

IP-protokollan tehtävä on siirtää tietosähkeitä yhdistettyjen verkkojen välillä. Tämä tapahtuu kuljettamalla tietosähkeitä yhdestä internetmoduulista toiseen, kunnes määränpää on saavutettu. Internetmoduulit sijaitsevat isäntäkoneessa ja yhdyskäytävät internetjärjestelmässä. (Postel 1981, 7.)

Tietosähkeet reititetään yhdestä internetmoduulista toiseen internetosoitteen avulla. Kun viestejä reititetään internetmoduulista toiseen, tietosähkeet voivat joutua kulkemaan verkon läpi, jonka paketin enimmäiskoko on pienempi kuin tietosähkeen koko.

Tämän ongelman ratkaisuun protokolla sisältää ominaisuuden sähkeiden paloittelemiseen. Tietosähkeen otsikon rakenne on esitelty kuviossa 1. (Postel 1981, 7.)



Kuvio 1. Tietosähkeen otsikon rakenne (mukaillen Postel 1981, 11.)

IP-protokollan lähetysjärjestelmä on epäluotettava, koska vastaanottajaan ei muodosteta yhteyttä. Tämän takia esimerkiksi monistettujen, kadonneiden tai väärässä järjestyksessä saapuvien tietosähkeiden virheenkorjaus ei onnistu. Nämä toiminnot voidaan kuitenkin toteuttaa muiden protokollien avulla. (Frystyk 1994.)

2.5 TCP-protokolla

TCP on yhteydellinen protokolla, eli lähetysjärjestelmä on luotettava. TCP muodostaa yhteyden kahden isäntäkoneen välille ennen kuin yhtään viestipakettia on lähetetty. Koska yhteys on muodostettu ennen viestipakettien lähetystä, voidaan varmistaa viestipakettien pääsy perille ja uudelleen lähettää matkalla kadonneet viestipaketit. TCP-protokollan ominaisuudet lisäävät viestijärjestelmän kuormitusta otsikon koon ja suuremman käsittelyajan takia. (Reynders & Wright 2003, 123.)

TCP-protokolla sisältää seuraavat toiminnot:

- Suurien viestilohkojen paloittelu pienempiin segmentteihin
- Datan uudelleenkokoaminen vastaanotetuista viestipaketeista
- Saapuvan viestin kuittaus
- Useiden yhteyksien muodostaminen etäkoneiden portteihin TCP-pistokepalveluilla
- Viestipakettien varmistus ja virheenhallinta
- Viestipakettien lähetyksen hallinta
- Viestipakettien jaksottelu ja uudelleenjärjestely. (Reynders & Wright 2003, 123.)

2.5.1 Portti

Siinä missä IP-osoite riittää viestin reitittämiseen tietylle koneelle internetissä, TCP-protokollan täytyy tietää, mille prosessille (eli ohjelmalle) kyseisellä koneella viesti on tarkoitettu. Kohdeprosessi määritellään porttinumeroilla 1–65535. (Reynders & Wright 2003, 124.)

2.5.2 Pistoke

Jotta voidaan määritellä sekä sijainti ja ohjelma, jolle tietty paketti kuuluu lähettää IP-osoite (sijainti) ja porttinumero (prosessi) yhdistetään pistokkeen osoitteeksi. IP-osoite sisältyy IP-otsikkoon ja porttinumero TCP-otsikkoon. Viestien kuljetus TCP-protokollan avulla onnistuu vain, jos pistoke löytyy sekä viestin lähteestä ja kohteesta. TCP myös mahdollistaa useiden pistokkeiden luomisen samaan porttiin. (Reynders & Wright 2003, 124.)

2.5.3 Sekvenssinumero

TCP-protokollan ytimeen kuuluu, että jokainen tavu TCP-yhteyden yli lähetettävää dataa sisältää uniikin 32-bittisen sekvenssinumeron. Jokaisen segmentin ensimmäisen tavun sekvenssinumero sisältyy viestin TCP-otsikkoon ja jokaiseen seuraavaan tavuun sekvenssinumeroa kasvatetaan, jotta vastaanottaja pystyy järjestelemään tavut. (Reynders & Wright 2003, 124.)

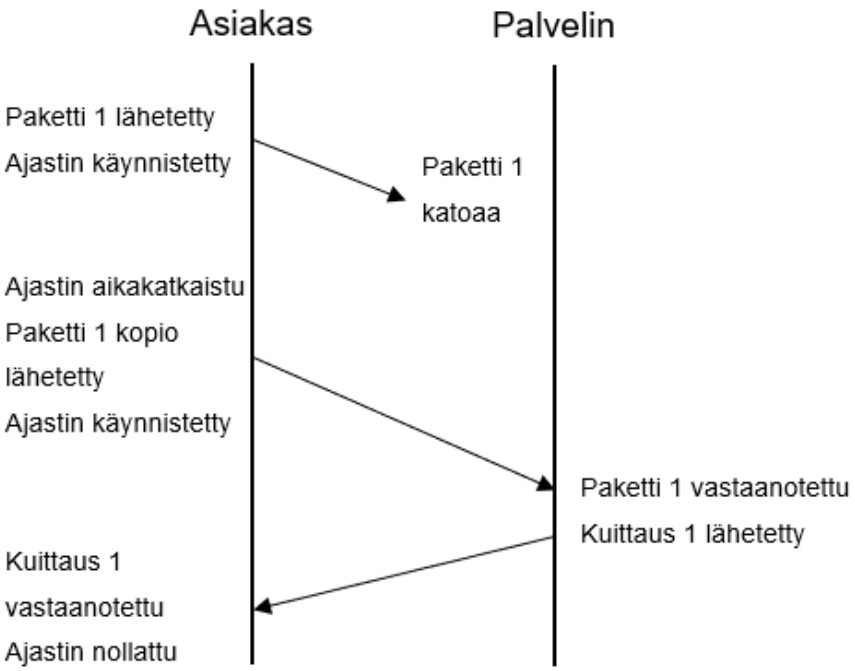
2.5.4 Kuittausnumero

TCP kuittaa vastaanotetun datan segmenteittäin, kuitenkin useita peräkkäisiä segmenttejä voidaan kuitata yhtä aikaa. Lähettäjälle palautunut kuittausnumero muodostuu viimeisen tavun numerosta, johon lisätään +1. Tällöin kuittausnumero osoittaa seuraavaan odotettuun sekvenssinumeroon. Jos jokin segmentti sisältää virheitä, kuittausnumero osoittaa virheellisen segmentin ensimmäiseen tavuun, jolloin

koko segmentti ja sitä seuraavat segmentit on lähetettävä uudelleen. (Reynders & Wright 2003, 125.)

2.5.5 Liukuva ikkuna

Segmentin perille pääsyn varmistamiseksi tarvitaan kuittausjärjestelmä. TCP-protokollan kuittausjärjestelmässä vastaanottaja lähettää kuittausviestin onnistuneesti saapuneista segmenteistä. Lähettäjä käynnistää ajastimen ja odottaa vastaanottajan kuittausviestiä tietyn ajan. Jos vastaanottaja ei vastaa tietyn ajan kuluttua, viestin kopio lähetetään vastaanottajalle. Kuittauksen toimintaperiaate näkyy kuviossa 2. (Reynders & Wright 2003 125.)

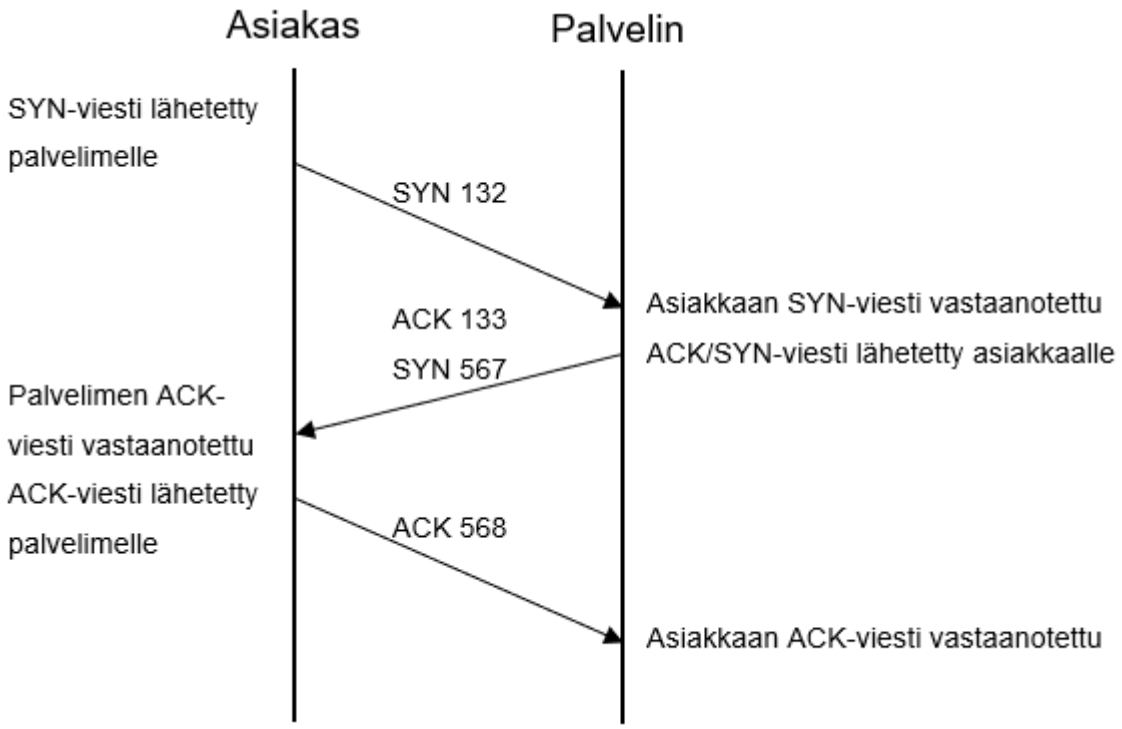


Kuvio 2. Kuittauksen toimintaperiaate (mukaillen Reynders & Wright 2003, 126.)

TCP käyttää positiiviseen kuittaukseen liukuvaa ikkunaa, koska yksittäisten kuittausten odottaminen jokaiselta lähetetyltä paketilta kuluttaisi paljon aikaa. Ideana on lähettää niin monta pakettia (tavua) kuin ikkunaan mahtuu, ennen kuin lähettäjä saa kuittauksen ensimmäiseen viestiin. Sitä mukaa kun kuittaukset saapuvat, ikkuna liukuu eteenpäin ja seuraava paketti voidaan lähettää. (Reynders & Wright 2003, 126.)

2.5.6 Yhteyden muodostaminen

Kaksisuuntaisen TCP-yhteyden muodostaminen tapahtuu kolmiosaisena SYN -> ACK/SYN -> ACK -kättelynä. Tämä prosessi näkyy kuviossa 3.



Kuvio 3. TCP-yhteyden muodostaminen (mukaillen Reynders & Wright 2003, 127.)

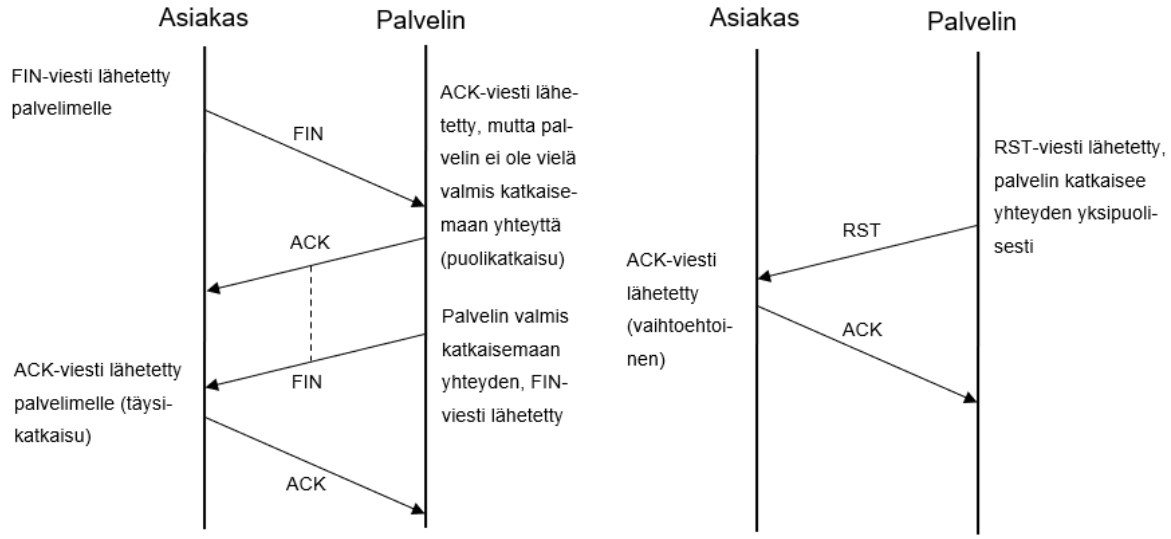
Yhteyden muodostava kone (asiakas) asettaa generoidun sekvenssinumeron kyseiseen kenttään TCP-kehyksessä ja asettaa SYN-lipun arvoon 1. Jos sekvenssinumero oli esimerkiksi 132, lähetetään viesti SYN 132 vastaanottajalle. (Reynders & Wright 2003, 127.)

Vastapuolen kone (palvelin) kuittaa yhteyden muodostuspyynnön kasvattamalla vastaanotettua sekvenssinumeroa yhdellä, asettamalla ACK-lipun arvoon 1 ja lähettämällä sen takaisin kuittausnumerona. Tässä tapauksessa kuittausviesti olisi ACK 133. Samaan aikaan palvelin asettaa oman sekvenssinumeronsa kehykseen ja asettaa SYN-lipun arvoon 1. Asiakkaalle lähetettävä viesti olisi esimerkiksi SYN 567. (Reynders & Wright 2003, 127.)

Yhteyden muodostava kone vastaanottaa viestin ja toteaa oman pyynnön kuitatuksi ja lähettää kuittauksen vastapuolen pyyntöön, tässä tapauksessa ACK 568. Näin kaksisuuntainen yhteys on muodostettu. (Reynders & Wright 2003, 127.)

2.5.7 Yhteyden sulkeminen

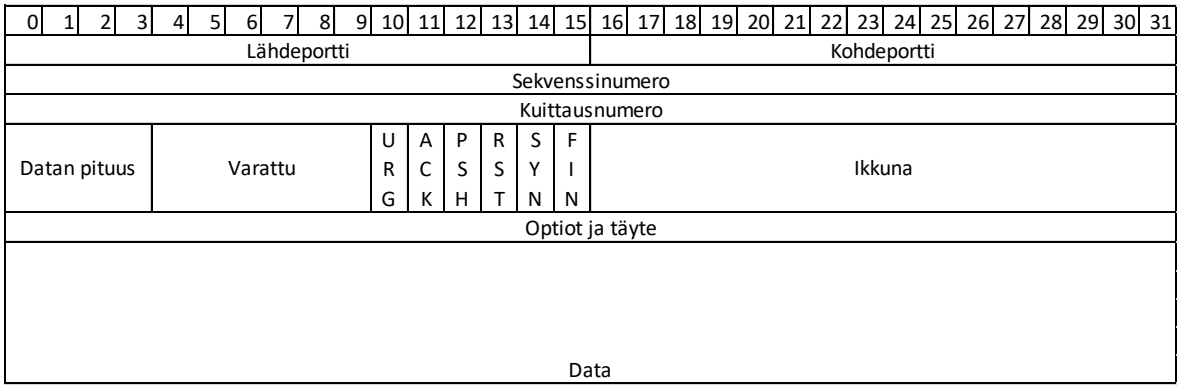
TCP-yhteyden sulkemiseen on eri tapoja. Yksi tapa on, että toinen osapuolista pyytää yhteyden sulkemista asettamalla otsikon FIN-lipun arvoon 1. Vastapuoli kuittaa tämän, mutta sen ei tarvitse katkaista yhteyttä heti, vaan se voi tarvittaessa lähettää lisää dataa (puolikatkaisu). Kun vastapuoli on valmis sulkemaan yhteyden, se lähettää sulkupyynnön, johon toinen osapuoli vastaa kuittausviestillä, ja yhteys on katkaistu (täysikatkaisu). Toinen tapa on, että toinen osapuolista katkaisee yhteyden yksipuolisesti asettamalla otsikon RST-lipun arvoon 1. Vastapuoli lähettää kuittauksen vaihtoehtoisesti ja katkaisee yhteyden. Esimerkit molemmista näkyy kuviossa 4. (Reynders & Wright 2003, 128.)



Kuvio 4. TCP-yhteyden katkaisu (mukaillen Reynders & Wright 2003, 128.)

2.5.8 TCP-kehys

TCP-kehys koostuu otsikosta ja datasta. Kehyksen rakenne näkyy kuviossa 5 ja otsikon rakenne taulukossa 2.



Kuvio 5. TCP-kehyksen rakenne (mukaillen Reynders & Wright 2003, 129.)

Taulukko 2. TCP-otsikon rakenne (mukaillen Reynders & Wright 2003, 129-130.)

Kentän nimi	Bittien lukumäärä	Kentän kuvaus
Lähdeportti (source port)	16	Lähdeportin numero.
Kohdeportti (destination port)	16	Kohdeportin numero.
Sekvenssinumero (sequence number)	32	Datan ensimmäisen tavun sekvenssinumero. Yhteyttä muodostettaessa käytetään alkuperäistä sekvenssinumeroa (ISN), jolloin seuraavan datan sekvenssinumero on ISN+1.
Kuittausnumero (acknowledgement number)	32	Jos ACK-lippu on 1, tämä kenttä sisältää sekvenssinumeron, jota tämän viestin lähettäjä odottaa seuraavaksi. Kun yhteys on muodostettu, kehys sisältää aina tämän kentän.

Datan pituus (data offset)	4	Kehyksen 32 bittisten sanojen lukumäärä. Tämä luku kertoo, mistä kehyksen data alkaa.
Varattu (reserved)	6	Varattu tulevaa käyttöä varten. Kentän bittien on oltava 0.
Liput (control bits/flags)	6	URG: Kiireellisen datan osoitin käytössä. ACK: Kuittausnumero käytössä. PSH: Datan välityksen pakotus käytössä. RST: Yhteyden katkaisu. SYN: Yhteyden muodostus. FIN: Yhteyden katkaisupyyntö.
Tarkistussumma (checksum)	16	Kenttä sisältää kehyksen ja pseudo-otsikon perusteella lasketun tarkistussumman.
Ikkuna (window)	16	Kenttä kertoo vastaanottajalle, kuinka paljon dataa lähettäjä pystyy vastaanottamaan.
Kiireellisen datan osoitin (urgent pointer)	16	Kiireellinen data on asetettu kehyksen alkuun ja tämä osoitin kertoo, mikä on viimeinen tavu kiireellistä dataa. Tätä kenttää tulkitaan vain, jos URG-lippu on 1.
Optiot ja täyte (options and padding)	0-320	Optiot voivat viedä tilaa otsikon lopusta, ja yhden option pituus on 8 bittiä. Jos käytettyjen optioiden bittien lukumäärä ei ole jaollinen 32:lla, lisätään perään 0-täytebittejä.

3 KÄYTETYT TYÖKALUT

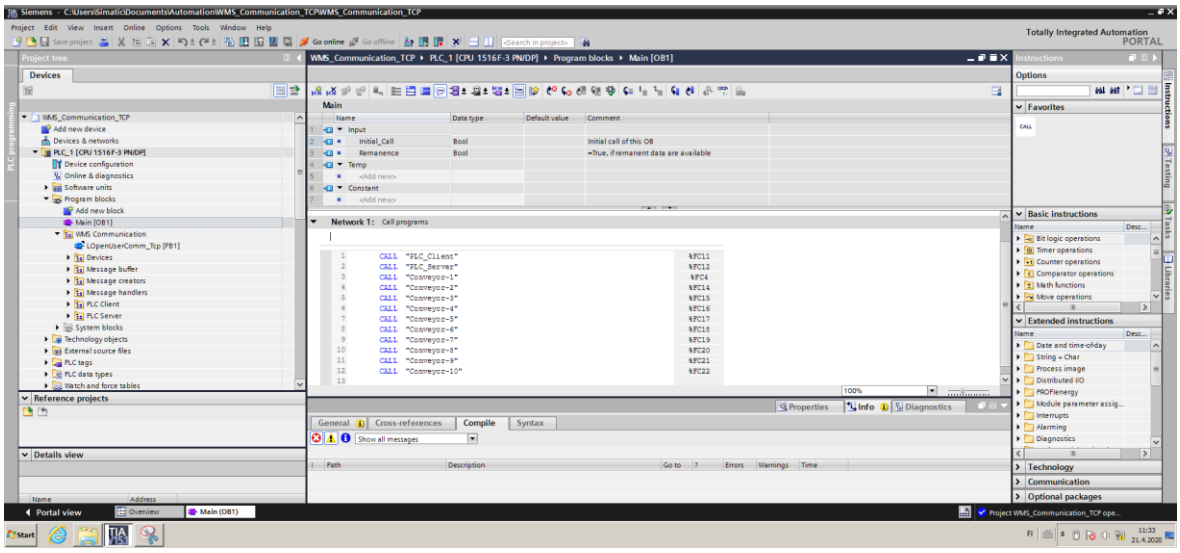
3.1 Siemens SIMATIC S7-1500 ohjelmoitava logiikka

SIMATIC S7-1500 -sarjan ohjelmoitavat logiikat kuuluvat Siemensin teollisen automaation tuotteisiin. Logiikan suorittimet skaalautuvat tehon, muistin ja määrän rakenteen suhteen, ne tarjoavat tiedonsiirron OPC UA- ja PROFINET-protokollan avulla sekä perustoiminnot liikkeenhallinnalle. Logiikan ominaisuuksia voidaan myös laajentaa erilaisilla moduuleilla. Kuviossa 6 on esitelty S7-1500-sarjan logiikoita. (Siemens 2020a.)

Kuvio 6. Siemens SIMATIC S7-1500 -sarjan PLC (Siemens 2020a.)

3.2 Siemens TIA Portal -suunnitteluohjelmisto

TIA Portal -suunnitteluohjelmisto kokoaa yhteen ohjelmistoon Siemensin perinteiset automaatio-ohjelmistot (STEP 7, WinCC, SINAMICS Startdrive, SIMOCODE ES ja SIMOTION SCOUT TIA). STEP 7 -ohjelmiston projektinäkymä löytyy kuviosta 7. (Siemens 2020b.)



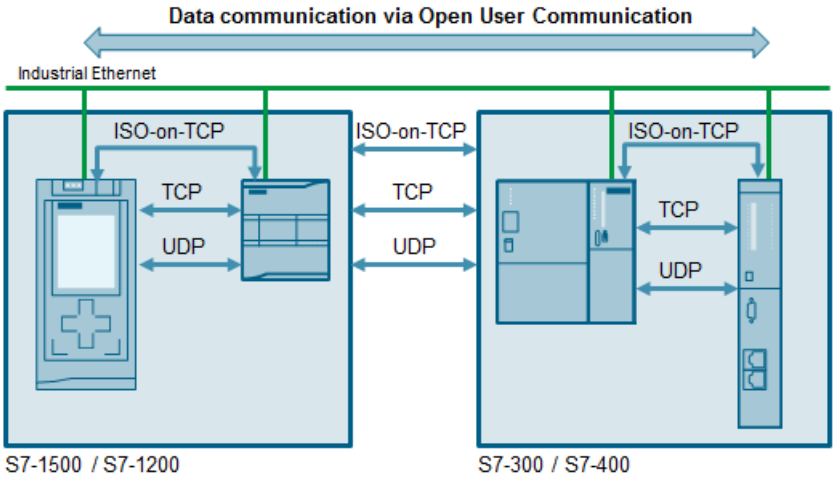
Kuvio 7. Siemens STEP 7 -ohjelmisto

Logiikan ohjelmointiin käytetty STEP 7 -ohjelmisto tarjoaa tuen seuraaville IEC-ohjelmointikielille:

- LAD (tikapuukaavio)
- FBD (toimintalohkokaavio)
- SCL (käskylista). (Siemens 2020b.)

3.3 Siemens LOpenUserComm-kirjasto

S7-1500-sarjan logiikoilla avoimen rajapinnan kommunikointi onnistuu käyttämällä TCON-, TSEND-, TRCV- ja TDISCON-funktioita. Funktioita käytettäessä on ohjelmoijan kuitenkin kirjoitettava niille oikeat parametrit ja huolehdittava virheenkäsittelystä. Siemensin sivuilta voi kuitenkin ladata LOpenUserComm-kirjaston, joka sisältää valmiit funktiolohkot. Kirjasto tukee TCP-, UDP- ja ISO-on-TCP-protokollia. Kirjaston tuetut kommunikointiprotokollat näkyvät kuviossa 8. (Siemens 2020c.)

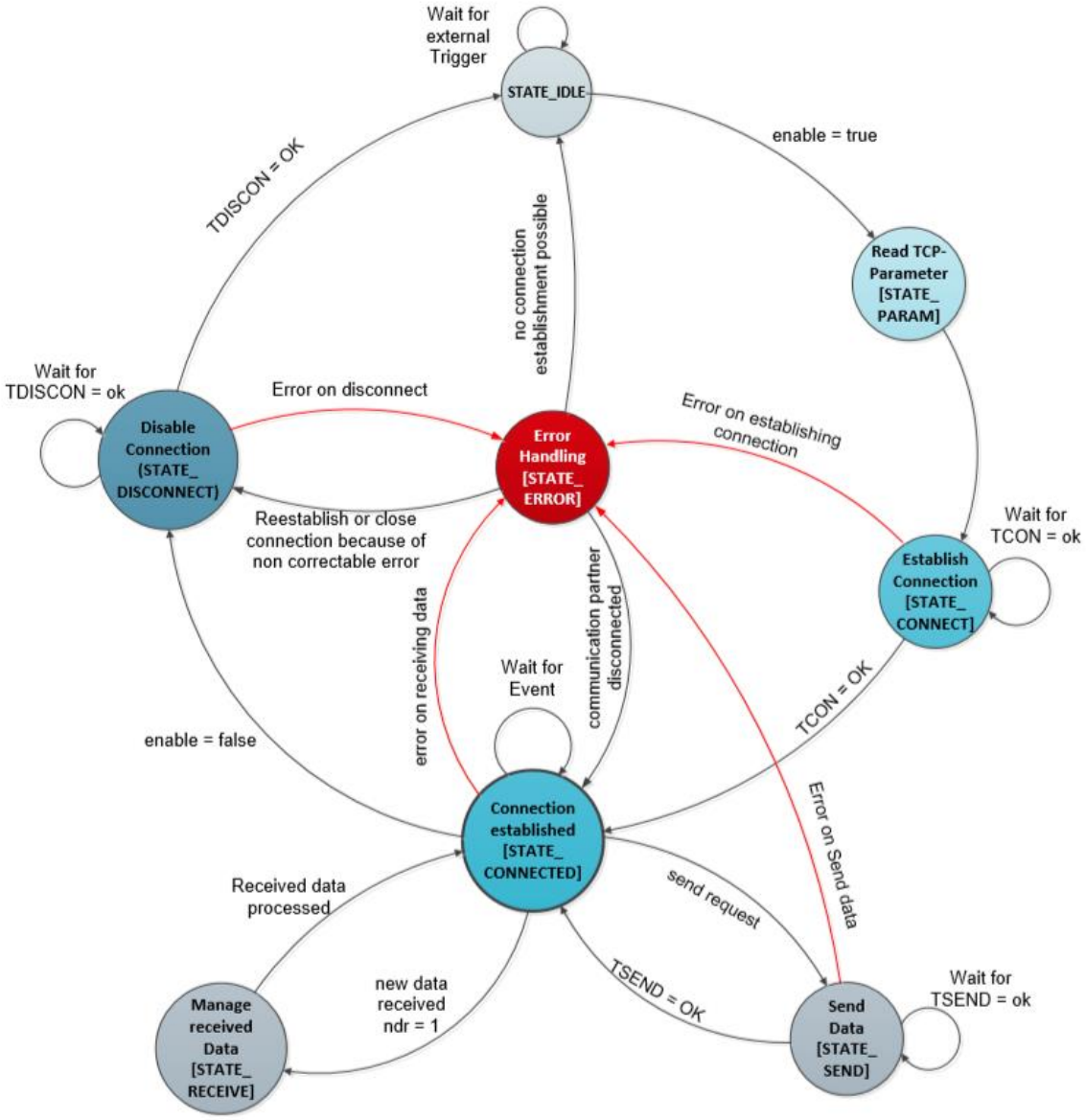


Kuvio 8. Avoimen rajapinnan kommunikointi (Siemens 2020c.)

Kirjaston kommunikoinnin funktiolohkot sisältävät seuraavat toiminnallisuudet:

- Yhteyden hallinta
- Datan lähetys toiselle laitteelle
- Datan vastaanottaminen toiselta laitteelta. (Siemens 2020c.)

Kirjastossa kommunikoinnin hallinta on toteutettu tilakoneena. Tilakoneen tilaa ajetaan syklisesti, kunnes siirtymisehto täyttyy. Tämä toteutus selkeyttää ohjelman logiikkaa ja helpottaa vianetsintää. Tilakoneen toimintaperiaate esitellään kuviossa 9 ja tilat taulukossa 3. (Siemens 2020d.)



Kuvio 9. Kommunikoinnin hallinnan tilakone (Siemens 2020d.)

Taulukko 3. Tilakoneen tilat (mukaillen Siemens 2020d.)

Tila	Kuvaus	Siirtymisehto
STATE_IDLE (1)	Yhteyksiä ei ole aktiivisena. Tilamuuttujat on resetoitu.	Jos yhdistä-parametri on aktivoitu, siirrytään seuraavaan tilaan.
STATE_PARAM (2)	Yhteyden parametrit luetaan ja aktivoidaan TCON-käsky.	Siirrytään suoraan seuraavaan tilaan ilman ehtoa.
STATE_CONNECT (3)	Yritetään muodostaa yhteys.	Jos yhteys ei ole muodostettu 180 sekunnissa, siirrytään tilaan STATE_ERROR. Jos yhteys on muodostettu, siirrytään tilaan STATE_CONNECTED.
STATE_DISCONNECT (4)	Tila sulkee yhteyden, jos katkaise yhteys -parametri on aktivoitu tai yhteyden sulkemista pyydetään uudelleenyhdistyksen tai virheen seurauksena.	Jos yhteys katkaistaan ilman virhettä, siirrytään tilaan STATE_IDLE. Jos yhteyden katkaisussa tapahtuu virhe, siirrytään tilaan STATE_ERROR.
STATE_SEND (5)	Aktivoi TSEND-käskyn parametrit. Ottaa TRCV-käskyn pois käytöstä. Odottaa, kunnes TSEND-käsky on valmis.	Jos lähetys onnistuu, siirrytään tilaan STATE_CONNECTED. Jos lähetyksen aikana tapahtuu virhe, siirrytään tilaan STATE_ERROR.

STATE_RECEIVE (6)	Tila prosessoi saapuvan datan. Voidaan määritellä määrämittaisille tai dynaamisille viesteille.	Kun saapunut data on prosessoitu, siirrytään tilaan STATE_CONNECTED.
STATE_CONNECTED (7)	Odottaa liipaisua lähetystilalta datan lähettämiseen. Varmistaa, onko vastapuoli vastaanottanut viestin. Valvoo yhteyden tilaa.	Jos datan vastaanottamisen aikana tapahtuu virhe, siirrytään tilaan STATE_ERROR. Jos yhteyden katkaisupyyntö on aktiivinen, siirrytään tilaan STATE_DISCONNECT. Jos datan lähetyspyyntö on aktiivinen, siirrytään tilaan STATE_SEND.
STATE_ERROR (8)	Tila päättää autonomisesti yritetäänkö virhe korjata siirtymällä muihin tiloihin. Toimittaa virheen tiedot lähtöparametreihin.	Jos yhteys pitää muodostaa uudelleen tai tapahtuu virhe, jota ei pystytä korjaamaan, siirrytään tilaan STATE_IDLE. Jos vastapuoli katkaisee yhteyden, siirrytään tilaan STATE_CONNECTED.

3.4 Java-ohjelmointikieli

Java on Sun Microsystemsin vuonna 1995 julkaisema yleiskäyttöinen ja alustariippumaton oliopohjainen ohjelmointikieli ja alusta. Alusta koostuu Java-virtuaalikoneesta (JVM) ja Javan ohjelmointirajapinnasta (API). Kuviossa 10 näkyy klassisen ”Hello World” -esimerkkiohjelman lähdekoodi Java-kielellä. (Greanier 2004, 3.)

```
package app;

public class App {
    public static void main(String[] args) throws Exception {
        System.out.println("Hello World!");
    }
}
```

Kuvio 10. ”Hello World” -esimerkkiohjelman lähdekoodi Java-kielellä

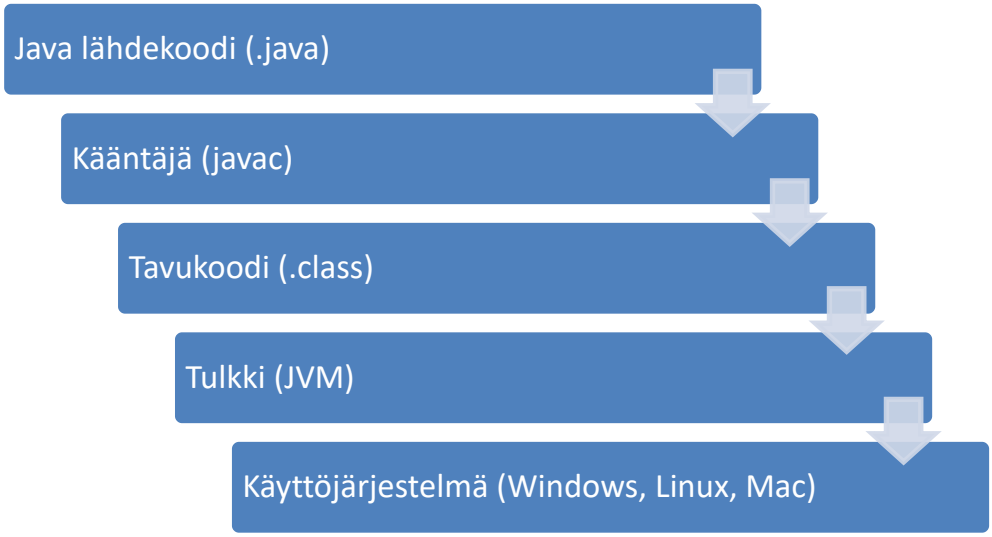
```
public class App {
    public App();
      Code:
        0: aload_0
        1: invokespecial #1                  // Method java/lang/Object."<init>":()V
        4: return

    public static void main(java.lang.String[]) throws java.lang.Exception;
      Code:
        0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
        3: ldc           #3                  // String Hello World!
        5: invokevirtual #4                  // Method ja-va/io/PrintStream.println:(Ljava/lang/String;)V
        8: return
}
```

Kuvio 11. ”Hello World” -esimerkkiohjelman purettu tavukoodi

Java-lähdekoodi välitetään kääntäjälle, joka generoi tavukoodin. Tavukoodia ei ole kohdistettu mihinkään tiettyyn alustaan. Sen sijaan JVM tulkitsee tavukoodin suorituksen aikana ja suorittaa sen. Tämä tarkoittaa, että vain JVM itse on alustasta riippuvainen, jolloin Java-ohjelmien tavukoodi pysyy alustasta riippumattomana. (Greanier 2004, 5.)

Kuviossa 11 näkyy ”javap -c”-komennolla esimerkkiohjelmasta purettu tavukoodi ja kuviossa 12 näkyy eri vaiheet lähdekoodista käyttöjärjestelmässä ajettavaksi ohjelmaksi.

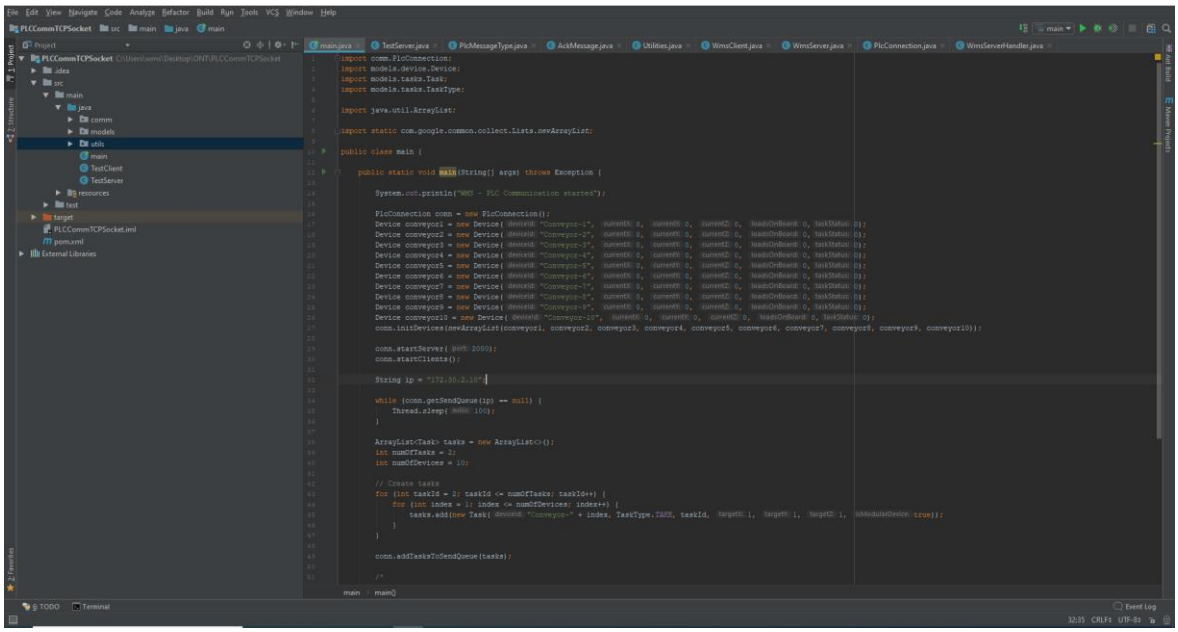


Kuvio 12. Prosessikaavio lähdekoodista ajettavaksi ohjelmaksi (mukaillen Gosling & McGilton 1995, 58.)

3.5 IntelliJ IDEA -ohjelmointiympäristö

IntelliJ IDEA on JetBrainsin kehittämä ohjelmointiympäristö (IDE) Java-kielisien ohjelmien kehittämiseen. Ohjelmointiympäristö analysoi lähdekoodin etsimällä yhteyksiä symbolien välillä kaikkien projektitiedostojen ja kielten välillä. Näiden tietojen avulla se mahdollistaa nopean navigoinnin, lähdekoodin virheanalyysin ja sen kätevän korjaamisen. (JetBrains [Viitattu 13.4.2020].)

Ympäristö sisältää myös useita kehittäjätyökaluja, kuten tuen useimmille versionhallintajärjestelmille, käännössovelluksille, testikehyksille, komentorivipäätteille ja sovelluspalvelimille. Ohjelmointiympäristön yleisnäkymä löytyy kuviosta 13. (JetBrains [Viitattu 13.4.2020].)



Kuvio 13. IntelliJ IDEA -ohjelmointiympäristö

4 TESTISOVELLUKSEN SUUNNITTELU

4.1 Sovelluksen vaatimukset

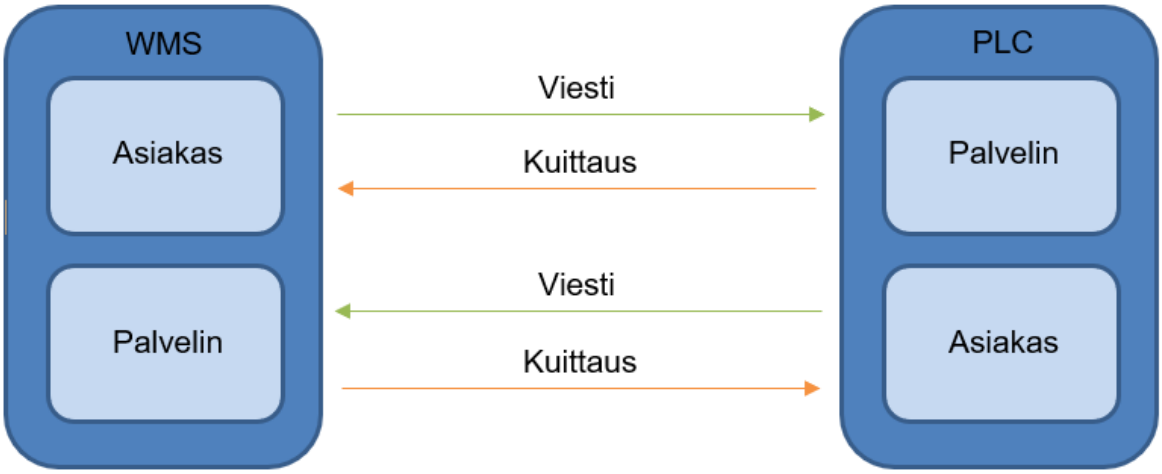
Testisovellukselle asetettiin seuraavat vaatimukset:

- WMS-varastonhallintajärjestelmän täytyy kommunikoida useiden PLC-logiikoihin yhdistettyjen laitteiden kanssa, esim. vaunujen ja kuljettimien.
- WMS:n ja PLC:n välinen yhteys toteutetaan TCP-pistokkeilla.
- Laitekommunikointi tapahtuu viesteillä, joilla on ennalta määritelty rakenne.
- Viestin vastaanottajan täytyy lähettää kuittaus viestin saapumisen vahvistamiseksi.
- Viestiä lähetetään uudelleen, kunnes saadaan kuittaus.
- Kommunikoinnin tulee olla mahdollisimman nopea ja luotettava suurella määrällä liitettyjä laitteita.

4.2 Sovelluksen kuvaus

Testisovellus koostuu PC:llä ajettavasta WMS-ohjelmasta sekä PLC-ohjelmasta. WMS- ja PLC-ohjelmat sisältävät asiakas- ja palvelintoteutuksen viestien lähettämiseen sekä vastaanottamiseen. Sovelluksen kuvaus näkyy kuviossa 14.

WMS-ohjelma lähettää sen tietokannasta löytyviä tehtäviä laitteille, jotka on kytketty PLC:hen. PLC-ohjelma lähettää sen viestipuskuriin kerättyjä viestejä WMS-järjestelmälle. Vastapuoli vastaa viestiin aina omalla kuittausviestillä. Yhteen WMS-ohjelmaan voi olla kytketty monta PLC-ohjelmaa.



Kuvio 14. Testisovelluksen kuvaus

4.3 Viestien rakenne

Laitekommunikoinnissa käytetyillä viesteillä on oltava ennalta määritelty rakenne, jotta viestin lukeminen PLC:n tietorakenteen muuttujiin tai sen sarjallistaminen Java-objekteiksi WMS-ohjelmassa onnistuisi. Oikeassa toteutuksessa näitä viestirakenteita olisi useita, mutta tässä testisovelluksessa käytetään vain kolmea. Viestien kentät esitellään taulukoissa 4, 5 ja 6.

4.3.1 ModularDeviceTaskWmsToPlc-viesti

ModularDeviceTaskWmsToPlc-viesti sisältää laitteelle annetun tehtävän parametreineen. Viestin lähetyssuunta on WMS → PLC.

Taulukko 4. Laitteelle lähetettävän tehtävän viestin kentät

Kenttä	Tyyppi	Sijainti tavutaulukossa	Tavujen lukumäärä
TCP-viestin pituus	Kokonaisluku	0	4
TCP-viestin tyyppi	Kokonaisluku	4	4

Laitteen tunnus	Merkkijono	8	30
Viestin numero	Kokonaisluku	38	4
Tehtävän numero	Kokonaisluku	42	4
Operaatiotyyppi	Kokonaisluku	46	4
Kohde	Kokonaisluku	50	4
Käsiteltävien kappaleiden määrä	Kokonaisluku	54	4
Kappaleen leveys 1…10	Kokonaisluku	58–94	4
Kappaleen korkeus 1…10	Kokonaisluku	98–134	4

4.3.2 ModularDeviceTaskPlcToWms-viesti

ModularDeviceTaskPlcToWms-viesti sisältää laitteelle annetun tehtävän tämänhetkisen tilan. Viestin lähetyssuunta on PLC → WMS.

Taulukko 5. Laitteelta lähetettävän tehtävän tilan viestin kentät

Kenttä	**Tyyppi**	**Sijainti tavutaulukossa**	**Tavujen lukumäärä**
TCP-viestin pituus	Kokonaisluku	0	4
TCP-viestin tyyppi	Kokonaisluku	4	4
Laitteen tunnus	Merkkijono	8	30
Viestin numero	Kokonaisluku	38	4
Tehtävän numero	Kokonaisluku	42	4

Viestin tyyppi	Kokonaisluku	46	4
Kohde	Kokonaisluku	50	4
Käsiteltävien kappaleiden määrä	Kokonaisluku	54	4
Virhekoodi	Kokonaisluku	58	4

4.3.3 AckMessage-viesti

AckMessage-viestiä käytetään vastaanotetun viestin kuittaamiseen ja se sisältää laitteen tunnuksen ja kuitattavan viestin numeron.

Taulukko 6. Kuittausviestin kentät

Kenttä	**Tyyppi**	**Sijainti tavutaulukossa**	**Tavujen lukumäärä**
TCP-viestin pituus	Kokonaisluku	0	4
TCP-viestin tyyppi	Kokonaisluku	4	4
Laitteen tunnus	Merkkijono	8	30
Kuitattavan viestin numero	Kokonaisluku	38	4

5 TESTISOVELLUKSEN WMS-OHJELMA

5.1 Yhteyden muodostaminen PLC-ohjelmaan

WMS-ohjelman palvelin on toteutettu Javan ServerSocket-luokan oliolla, joka sisältää palvelinpistokkeen toiminnot. ServerSocket-luokan konstruktori ottaa parametrina portin numeron, josta palvelin kuuntelee yhteyspyyntöjä. WMS-palvelinluokka toteuttaa Runnable-rajapinnan ja sen run-metodin, jota kutsumalla voidaan ajaa metodin sisältämää työtä omassa säikeessä.

```
public class WmsServer implements Runnable {

    private PlcConnection conn;
    private final int port;

    public WmsServer(int port, PlcConnection conn) {
        this.conn = conn;
        this.port = port;
    }

    @Override
    public void run() {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            while (true) {
                Socket clientSocket = serverSocket.accept();
                conn.startServerHandler(clientSocket, conn);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Kuvio 15. WmsServer-luokka

Kuviossa 15 näkyy WMS-palvelinluokka, jonka run-metodin työkierto on seuraavanlainen:

Ensiksi yritetään käynnistää palvelinpistoke kuuntelija parametrina saatuun porttiin. Jos tapahtuu virhe, se tulostetaan konsoliin ja yritetään uudelleen seuraavalla työkierrolla.

Sen jälkeen siirrytään kuuntelusilmukkaan, joka odottaa saapuvaa yhteyspyyntöä. Kun yhteyspyyntö saapuu, palauttaa accept-metodi Socket-luokan olion, joka sisältää pistokeyhteyden asiakkaaseen.

Seuraavaksi kutsutaan PlcConnection-luokan startServerHandler-metodia antaen sille parametreina Socket- ja PlcConnection-oliot. Metodi käynnistää asiakasyhteydelle käsittelijän omaan säikeeseen. Sitten siirrytään takaisin silmukan alkuun odottamaan uutta yhteyspyyntöä. StartServerHandler-metodi näkyy kuviossa 16.

```
public void startServerHandler(Socket clientSocket, PlcConnection conn) {
    WmsServerHandler wmsServerHandler = new WmsServerHandler(clientSocket, conn);
    executorService.execute(wmsServerHandler);
}
```

Kuvio 16. PlcConnection-luokan startServerHandler-metodi

WMS-ohjelman asiakassäikeen käynnistäminen tapahtuu kutsumalla PlcConnection-luokan startClients-metodia. Metodi kutsuu WmsClient-luokan konstruktoria ja antaa sille parametreina PLC:n IP-osoitteen, porttinumeron, viiveen yhteyden uudelleen muodostamiseen sekä PlcConnection-luokan olion. Tämä näkyy kuviossa 17.

```
public void startClients() {
    deviceMap.forEach((device, props) -> {
        // Check if there is no client for device ip address
        if (!wmsClients.stream().anyMatch(client -> client.getIp().equals(props.getIp()))) {
            // Create client
            WmsClient wmsClient = new WmsClient(props.getIp(), props.getPort(), 10000, this);
            wmsClients.add(wmsClient);
            sendQueue.putIfAbsent(props.getIp(), new LinkedBlockingDeque<>());
            executorService.execute(wmsClient);
        }
    });
}
```

Kuvio 17. PlcConnection-luokan startClients-metodi

5.2 Viestin vastaanottaminen

Viestin vastaanottaminen on toteutettu WMS-ohjelmassa Javan Socket-luokan olilla. Socket-luokka sisältää getInputStream- ja getOutputStream-metodit, joiden avulla voidaan lähettää ja vastaanottaa dataa pistokeyhteyden yli.

```java
public class WmsServerHandler implements Runnable {

    private Socket clientSocket;
    private PlcConnection conn;
    private String ip;

    public WmsServerHandler(Socket clientSocket, PlcConnection conn) {
        this.clientSocket = clientSocket;
        this.conn = conn;
        this.ip = "";
    }

    @Override
    public void run() {
        try {
            DataInputStream input = new DataInputStream(clientSocket.getInputStream());
            DataOutputStream output = new DataOutputStream(clientSocket.getOutputStream());
            ip = (((InetSocketAddress) clientSocket.getRemoteSocketAddress()).getAddress()).toString().replace("/","");
            System.out.println("Server connected to " + ip);

            long start = 0;
            long end = 0;
            int numOfMsg = 0;

            while (true) {
                IPlcMessage message = readBytes(input);     // Get received message from PLC
                System.out.println(message.toString() + " read from " + message.getDeviceId() + " at " + ip);
                ackMessage(output, message);                // Handle message and ack

                numOfMsg++; // Increment number of messages received, testing

                // Timer for receiving 50 messages from PLC
                if (numOfMsg == 1) {
                    start = System.currentTimeMillis();
                } else if (numOfMsg == 50) {
                    end = System.currentTimeMillis();
                    System.out.println("Message cycle took: " + (end - start) + " ms" );
                }

                Thread.sleep(50); // Small delay between reading input
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Kuvio 18. WmsServerHandler-luokan konstruktori ja run-metodi

Kuviosta 18 löytyy luokka, joka on palvelimen asiakasyhteyden käsittelijä. Kaikki työkierron vaiheet ovat try/catch-lohkon sisällä, joka ottaa kiinni ajon aikana tapahtuvat virheet ja tulostaa ne konsoliin.

Ensiksi luodaan DataInputStream- ja DataOutputStream-oliot TCP-viestien vastaanottamista ja lähettämistä varten pistokkeen getInputStream- ja getOutputStream-metodien avulla. Sen jälkeen tulostetaan konsoliin asiakkaan IP-osoite yhteyden testaamisen avuksi.

Seuraavaksi siirrytään viestien vastaanotto- ja kuittaussilmukkaan. Tavuvirtana saapuvasta TCP-viestistä muodostetaan viestiä vastaava Java-olio kuviossa 19 näky-

vällä readBytes-metodilla, joka saa parametrina pistokkeen vastaanotto-olion. Metodi kerää saapuvan tavuvirran tavutaulukkoihin, joista parsitaan viestin pituus, tyyppi ja data. Oikeassa sovelluksessa saapunut viesti lisättäisiin joko tietokantaan tai puskurijonoon käsittelyä varten.

```
private IPlcMessage readBytes(DataInputStream input) throws Exception {
    int offset = 0;
    int bytesRead = 0;

    // Parse tcpFrameLength from received message (bytes 0-3)
    byte[] tcpFrameLengthData = new byte[4];
    while ((bytesRead = input.read(tcpFrameLengthData, offset, 4 - offset)) != -1) {
        offset += bytesRead;
        if (offset >= 4) break;
    }
    int tcpFrameLength = ByteBuffer.wrap(tcpFrameLengthData).getInt();

    // Parse plcMessageType from received message (bytes 4-7)
    byte[] plcMessageTypeData = new byte[4];
    while ((bytesRead = input.read(plcMessageTypeData, 4 - offset, 8 - offset)) != -1) {
        offset += bytesRead;
        if (offset >= 8) break;
    }
    PlcMessageType plcMessageType = PlcMessageType.fromValue(ByteBuffer.wrap(plcMessageTypeData).getInt());

    // Insert received bytes to byte array
    byte[] data = new byte[tcpFrameLength];
    data[0] = tcpFrameLengthData[0];
    data[1] = tcpFrameLengthData[1];
    data[2] = tcpFrameLengthData[2];
    data[3] = tcpFrameLengthData[3];

    data[4] = plcMessageTypeData[0];
    data[5] = plcMessageTypeData[1];
    data[6] = plcMessageTypeData[2];
    data[7] = plcMessageTypeData[3];

    // Read rest of the message
    while ((bytesRead = input.read(data, offset, data.length - offset)) != -1) {
        offset += bytesRead;
        if (offset >= data.length) break;
    }

    // Create correct message instance based on message type
    IPlcMessage message;
    switch (plcMessageType) {
        case ModularDeviceLoadReportPlcToWms:   message = new ModularDeviceLoadReportPlcToWms(); break;
        case ModularDeviceTaskPlcToWms:         message = new ModularDeviceTaskPlcToWms(); break;
        case ModularDeviceStatusPlcToWms:       message = new ModularDeviceStatusPlcToWms(); break;
        case DeviceTaskPlcToWms:                message = new DeviceTaskPlcToWms(); break;
        default: throw new Exception("Message type error");
    }

    return message.fromBytes(data);
}
```

Kuvio 19. WmsServerHandler-luokan readBytes-metodi

Kun viesti on luettu, lähetetään kuittausviesti kuviosta 20 löytyvällä ackMessage-metodilla, joka saa parametreina pistokkeen lähetysolion ja kuitattavan viestin. Metodi poimii vastaanotetusta viestistä laitetunnuksen, viestinumeron sekä viestityypin

ja kutsuu niillä AckMessage-luokan konstruktoria, joka palauttaa kuittausviestiä mallintavan olion. Olio muutetaan tavutaulukoksi sen toBytes-metodilla ja lähetetään asiakkaalle. Konsoliin tulostetaan lähetty kuittausviesti.

```
private void ackMessage(DataOutputStream output, IPlcMessage message) throws IOException {
    IPlcMessage ackMessage = new AckMessage(message.getDeviceId(), message.getMsgNumber(), message.getPlcMessageType());
    output.write(ackMessage.toBytes(), 0, ackMessage.getTcpFrameLength());
    output.flush();
    System.out.println(ackMessage.toString() + " ack sent to " + ip);
}
```

Kuvio 20. WmsServerHandler-luokan ackMessage-metodi

Kuvion 18 silmukan lopussa on testaamista varten tehty laskuri ja ajastin, joka tehtävä on tulostaa konsoliin, kuinka kauan 50 viestin vastaanottaminen asiakkaalta kestää. Tämän perusteella tehtiin arvio TCP-toteutuksen suorituskyvystä. Silmukan viimeisen rivin 50 ms viive ”Thread.sleep”-käskyllä lisättiin, koska ilman sitä viestien lukeminen ei toiminut oikein.

5.3 Viestin lähettäminen

Viestin lähettämiseen käytetään pistokkeen samoja metodeja kuin niiden vastaanottamiseen. Tämän toiminnon WmsClient-luokka ja sen run-metodin lähdekoodi on kuviossa 21.

```java
public class WmsClient implements Runnable {

    private PlcConnection conn;
    private final String ip;
    private final int port;
    private final int timeout;  // Reconnection delay in milliseconds

    public WmsClient(String ip, int port, int timeout, PlcConnection conn) {
        this.ip = ip;
        this.port = port;
        this.timeout = timeout;
        this.conn = conn;
    }

    @Override
    public void run() {
        while (true) {
            try (Socket serverSocket = new Socket(ip, port);
                 DataInputStream input = new DataInputStream(serverSocket.getInputStream());
                 DataOutputStream output = new DataOutputStream(serverSocket.getOutputStream()))
            {
                System.out.println("Client connected to " + ip + ":" + port);
                while (true) {
                    if (input.available() > 0) {
                        // Read ack
                        IPlcMessage ackMessage = readBytes(input);      // Get received ack message from PLC
                        conn.getSendQueue(ip).take();                   // Remove acked message from send queue
                        System.out.println(ackMessage.toString() + " read from " + ip);
                    } else if (conn.getSendQueue(ip).size() > 0) {
                        // Send task
                        sendTask(output);
                    }
                    Thread.sleep(50); // Small delay between reading input
                }
            } catch (Exception e) {
                e.printStackTrace();
            } finally {
                try {
                    System.out.println("Attempting to reconnect to address: " + ip + ":" + port + " in " + timeout/1000 + " seconds");
                    Thread.sleep(timeout);
                } catch (InterruptedException e) {}
            }
        }
    }
```

Kuvio 21. WmsClient-luokka ja run-metodi

Run-metodin yhteyden muodostaminen PLC:n palvelinohjelmaan tapahtuu silmukan sisällä, eli jos yhteyden muodostaminen ei onnistu, ohjelma tulostaa konsoliin virheilmoituksen ja yrittää uudelleen sille määritellyn ajan kuluttua.

Kun yhteyden muodostaminen onnistuu, siirrytään silmukkaan, jossa joko lähetetään viesti, odotetaan kuittausta tai pidetään säie odottavassa tilassa.

Tehtäväviestin lähettäminen PLC:hen kytketylle laitteelle tapahtuu kuviossa 22 olevalla sendTask-metodilla.

```
private void sendTask(DataOutputStream output) throws Exception {
    // Check if there is any tasks to send
    if (conn.getSendQueue(ip).isEmpty()) {
        return;
    }

    // Get message to send
    IPlcMessage taskMessage = conn.getSendQueue(ip).peek();

    // Send task
    output.write(taskMessage.toBytes(), 0, taskMessage.getTcpFrameLength());
    output.flush();
    System.out.println(taskMessage.toString() + " task sent to " + device.get().getDeviceId() + " at " + ip);
}
```

Kuvio 22. WmsClient-luokan sendTask-metodi

SendTask-metodi tarkistaa, löytyykö lähetysjonosta viestejä. Jos ei löydy, palataan takaisin silmukkaan. Jos jonossa on viestejä, käydään lukemassa viestijono peek-metodilla, joka palauttaa jonon ensimmäisen viestin, mutta ei poista sitä jonosta. Viesti muutetaan tavutaulukoksi toBytes-metodilla ja sen jälkeen se lähetetään PLC:n palvelimelle pistokkeen write-metodilla. Viestiä lähetetään uudelleen, kunnes kuittaus on saatu.

PLC:ltä saapuva kuittausviesti luetaan WmsClient-luokan readBytes-metodilla, joka näkyy kuviossa 23. Metodin toiminta on samankaltainen kuin WmsServerHandler-luokan readBytes-metodilla. Kun kuittausviesti on luettu, poistetaan lähetysjonon ensimmäinen viesti jonosta jonon take-metodilla.

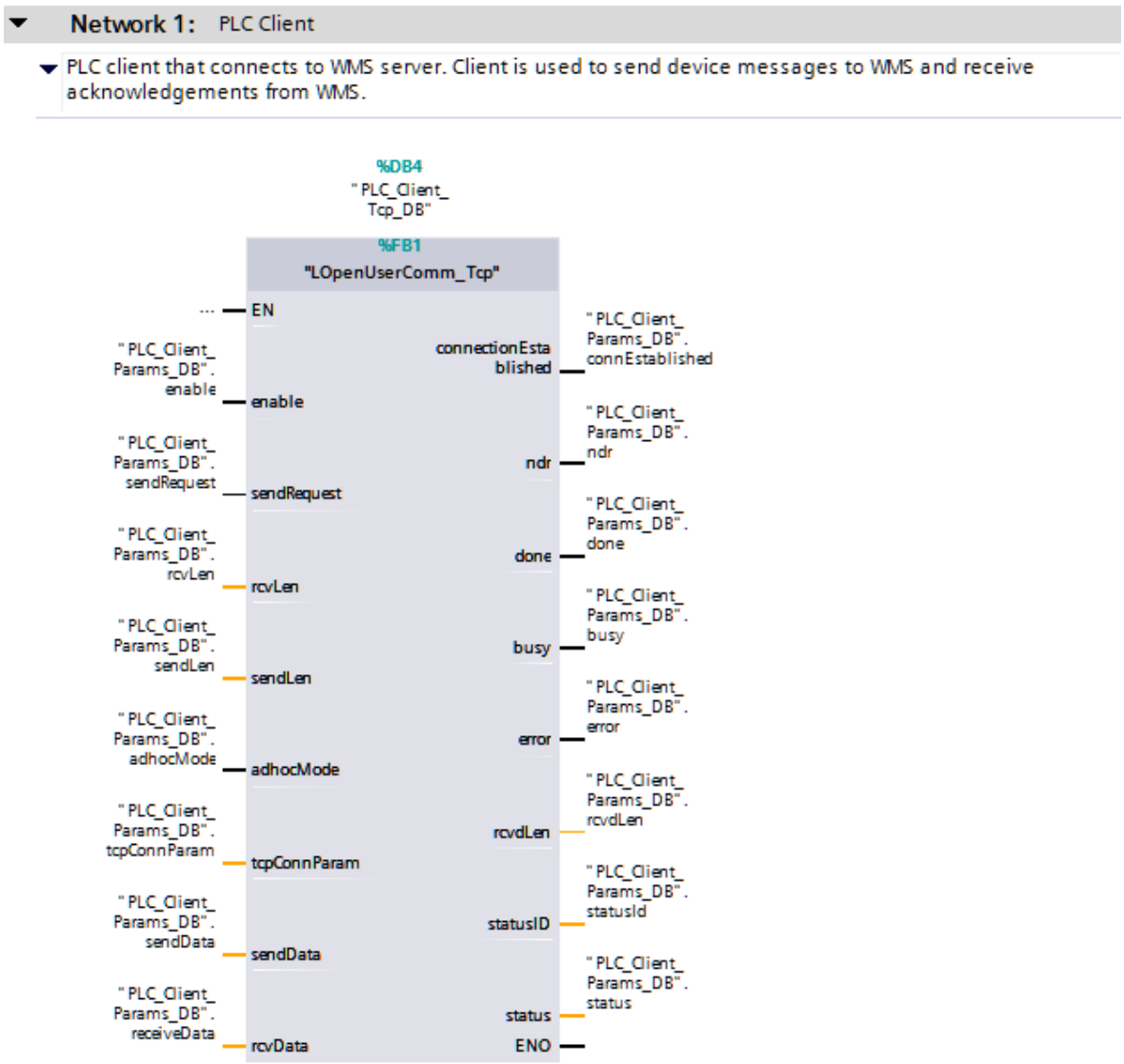
```
private IPlcMessage readBytes(DataInputStream input) throws Exception {
    IPlcMessage message = new AckMessage();
    int offset = 0;
    int bytesRead = 0;
    byte[] data = new byte[message.getTcpFrameLength()];
    while ((bytesRead = input.read(data, offset, data.length - offset)) != -1) {
        offset += bytesRead;
        if (offset >= data.length) break;
    }
    return message.fromBytes(data);
}
```

Kuvio 23. WmsClient-luokan readBytes-metodi

6 TESTISOVELLUKSEN PLC-OHJELMA

6.1 Yhteyden muodostaminen WMS-ohjelmaan

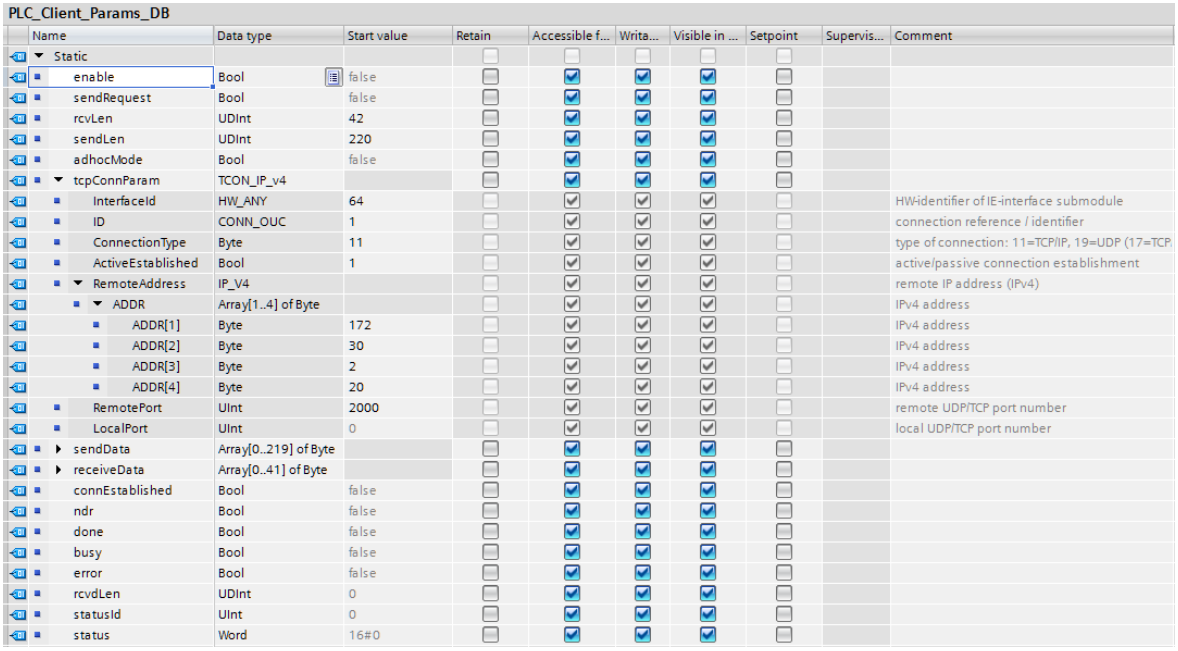
PLC:n asiakas- sekä palvelinohjelmat on toteutettu Siemensin LOpenUserComm_Tcp-kirjaston avulla. Kirjaston funktiolohko tarvitsee yhteysparametrit sen tcpConnParam-tuloon. Asiakasohjelman funktiolohko näkyy kuviossa 24. Palvelinohjelma on toteutettu samalla funktiolohkolla, paitsi parametrit ovat erilaiset. Asiakasohjelman parametrien asetukset ovat kuviossa 25 ja palvelimen kuviossa 26.



Kuvio 24. PLC:n asiakasohjelman funktiolohko

Asiakasohjelma yrittää muodostaa yhteyden parametreissa määritettyyn palvelimeen, kun sen enable-tulo on asetettu arvoon 1.

Palvelinohjelma kuuntelee yhteyspyyntöjä parametreissa määritellystä portista, kun sen enable-tulo on asetettu arvoon 1.



PLC_Client_Params_DB

Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Supervis...	Comment
Static									
enable	Bool	false		☑	☑	☑			
sendRequest	Bool	false		☑	☑	☑			
rcvLen	UDInt	42		☑	☑	☑			
sendLen	UDInt	220		☑	☑	☑			
adhocMode	Bool	false		☑	☑	☑			
tcpConnParam	TCON_IP_v4			☑	☑	☑			
InterfaceId	HW_ANY	64		☑	☑	☑			HW-identifier of IE-interface submodule
ID	CONN_OUC	1		☑	☑	☑			connection reference / identifier
ConnectionType	Byte	11		☑	☑	☑			type of connection: 11=TCP/IP, 19=UDP (17=TCP
ActiveEstablished	Bool	1		☑	☑	☑			active/passive connection establishment
RemoteAddress	IP_V4			☑	☑	☑			remote IP address (IPv4)
ADDR	Array[1..4] of Byte			☑	☑	☑			IPv4 address
ADDR[1]	Byte	172		☑	☑	☑			IPv4 address
ADDR[2]	Byte	30		☑	☑	☑			IPv4 address
ADDR[3]	Byte	2		☑	☑	☑			IPv4 address
ADDR[4]	Byte	20		☑	☑	☑			IPv4 address
RemotePort	UInt	2000		☑	☑	☑			remote UDP/TCP port number
LocalPort	UInt	0		☑	☑	☑			local UDP/TCP port number
sendData	Array[0..219] of Byte			☑	☑	☑			
receiveData	Array[0..41] of Byte			☑	☑	☑			
connEstablished	Bool	false		☑	☑	☑			
ndr	Bool	false		☑	☑	☑			
done	Bool	false		☑	☑	☑			
busy	Bool	false		☑	☑	☑			
error	Bool	false		☑	☑	☑			
rcvdLen	UDInt	0		☑	☑	☑			
statusId	UInt	0		☑	☑	☑			
status	Word	16#0		☑	☑	☑			

Kuvio 25. PLC:n asiakasohjelman parametrit

PLC_Server_Params_DB

Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Supervis...	Comment
Static									
enable	Bool	false							
sendRequest	Bool	false							
rcvLen	UDInt	0							
sendLen	UDInt	42							
adhocMode	Bool	true							
tcpConnParam	TCON_IP_v4								
InterfaceId	HW_ANY	64							HW-identifier of IE-interface submodule
ID	CONN_OUC	2							connection reference / identifier
ConnectionType	Byte	11							type of connection: 11=TCP/IP, 19=UDP (17=TCP
ActiveEstablished	Bool	0							active/passive connection establishment
RemoteAddress	IP_V4								remote IP address (IPv4)
ADDR	Array[1..4] of Byte								IPv4 address
ADDR[1]	Byte	172							IPv4 address
ADDR[2]	Byte	30							IPv4 address
ADDR[3]	Byte	2							IPv4 address
ADDR[4]	Byte	20							IPv4 address
RemotePort	UInt	0							remote UDP/TCP port number
LocalPort	UInt	3000							local UDP/TCP port number
sendData	Array[0..41] of Byte								
receiveData	Array[0..899] of Byte								
connEstablished	Bool	false							
ndr	Bool	false							
done	Bool	false							
busy	Bool	false							
error	Bool	false							
rcvdLen	UDInt	0							
statusId	UInt	0							
status	Word	16#0							

Kuvio 26. PLC:n palvelinohjelman parametrit

Kuvioissa 24 esitelty funktiolohko sisältää seuraavat tulot ja lähdöt:

- **enable:** Signaali yhteyden muodostamiseen ja datan vaihtoon.
- **sendRequest:** Liipaisu lähetystehtävälle.
- **rcvLen:** Vastaanotettavan data pituus. Ei ole käytössä, jos adhoc-tila on päällä.
- **sendLen:** Lähetystehtävän tavujen lukumäärä. Jos adhoc-tila on päällä, tavujen lukumäärä pitää lähettää neljässä ensimmäisessä tavussa.
- **adhocMode:** Jos tila on päällä, voidaan vastaanottaa eripituisia viestejä.
- **tcpConnParam:** Yhteysparametrit.
  - **InterfaceId:** Rajapinnan laitetunnus.
  - **ID:** Yhteyden tunnistenumero.
  - **ConnectionType:** Yhteystyyppi, 11 = TCP.
  - **ActiveEstablished:** Yhteyden muodostustyyppi. 0 = palvelin, 1 = asiakas.
  - **RemoteAddress:** Vastapuolen IP-osoite.
  - **RemotePort:** Vastapuolen porttinumero.
  - **LocalPort:** Oma porttinumero.

- **connectionEstablished:** Yhteyden tila.
- **ndr:** Vastaanotetaan uutta dataa.
- **done:** Lähetystehtävä valmis.
- **busy:** Funktiolohko käynnissä.
- **error:** Virheen tila.
- **rcvdLen:** Vastaanotetun datan tavujen lukumäärä.
- **statusID:** Virheen tunniste.
- **status:** Funktiolohkon tila.
- **sendData:** Lähetettävän datan alue.
- **rcvData:** Vastaanotettavan datan alue. (Siemens 2020d.)

6.2 Viestin vastaanottaminen

Palvelimen funktiolohko asettaa vastaanotetun viestin rcvData-alueelle tavutaulukkomuodossa. Viestin tyypin perusteella se siirretään sopivaan tietorakenteeseen Deserialize-funktion avulla. Tämä esitellään kuviossa 27. Deserialize-funktio tarvitsee kolme parametria. SRC_ARRAY on lähdetaulukko, josta data kopioidaan tavu kerrallaan POS-parametrin osoittamasta paikasta kohteeseen DEST_VARIABLE.

```
// Convert byte array data to TaskWMStoPLC data type
#deserializeResult := Deserialize(SRC_ARRAY := "PLC_Server_Params_DB".receiveData,
                                  DEST_VARIABLE => "Device_DB".ModularDevices[#modDevIndex].TaskWMStoPLC,
                                  POS := #Position);
```

Kuvio 27. Tavutaulukon muuttaminen tietorakenteeksi Deserialize-funktiolla

Vastaanotetusta viestistä kerätään tarvittavat arvot tavutaulukkoon, joka lähetetään kuittausviestinä. Kuittausviestin tavutaulukko asetetaan palvelimen funktiolohkon sendData-alueelle, josta se lähetetään asiakkaalle, kun sendRequest-tuloa on liipaistu.

6.3 Viestin lähettäminen

Viestit, joita halutaan lähettää PLC:ltä WMS-ohjelmalle kerätään viestipuskuriin tavutaulukkomuodossa. Viestin muuttaminen tavutaulukoksi tapahtuu Serialize-funktion avulla, joka toimii kuten Deserialize-funktio, paitsi tämä kopioi tietorakenteen tavu kerrallaan taulukkoon. Viestin muuttaminen tavutaulukoksi näkyy kuviossa 28.

```
// Convert message data to byte array
#serializeResult := Serialize(SRC_VARIABLE := #bytes,
                              DEST_ARRAY => #TaskPLCtoWMS.HEADER,
                              POS := #Position);
```

Kuvio 28. Tietorakenteen muuttaminen tavutaulukoksi Serialize-funktiolla

Lähetettävä data kerätään viestipuskuriin, josta asiakasohjelman funktiolohko käy määritellyn ajan välein tarkistamassa, onko lähetettäviä viestejä. Jos puskurissa on viestejä, asetetaan puskurin ensimmäinen viesti tavutaulukkomuodossa funktiolohkon sendData-alueelle ja liipaistaan sendRequest-tuloa. Viestiä lähetetään uudelleen, kunnes kuittaus on saatu. Kuittauksen saapuessa puskurin ensimmäinen viesti todetaan kuitatuksi ja siirrytään seuraavaan.

7 TULOKSET

Testisovelluksella ajettujen testien perusteella yhden viestin lähettäminen ja kuittauksen vastaanottaminen kestää noin 100 ms, eli lähetysnopeus on noin 50 ms per viesti. Testisovelluksella lähetettiin vain pelkkiä testiviestejä simuloiden kymmentä laitetta, eli se ei vastaa täysin varastohallintaohjelmiston tuotantoajoa, jossa WMS- ja PLC-ohjelmat suorittavat paljon muitakin raskaita toimintoja, ja käskytettäviä laitteita voi olla useita kymmeniä tai jopa satoja.

Jos kommunikointi toteutetaan tällä tavalla, ongelmaksi muodostuu se, että viestejä lähetetään vain yksi kerrallaan. Viestin lähetysnopeutta ei juurikaan pysty parantamaan, eli ainoaksi vaihtoehdoksi jää usean viestin kerääminen yhdeksi lähetettäväksi viestiksi. Tämä vaatisi suuria muutoksia viestien käsittelyyn, sekä se voisi heikentää PLC-ohjelman suorituskykyä.

Näiden havaintojen perusteella voidaan todeta, että TCP-pistokeyhteydellä toteutettu kommunikointi ei tuo haluttua suorituskykyparannusta nykyiseen varastonhallintajärjestelmän toteutukseen. TCP-pistokkeilla toteutettu yhteys voi kuitenkin sopia käyttötarkoitukseen, joka ei vaadi useita rinnakkain lähetettäviä viestejä.

8 YHTEENVETO JA POHDINTA

Tämän opinnäytetyön tavoitteena oli tutkia, onnistuuko Pesmelin WMS-varastonhallintajärjestelmän laitekommunikoinnin toteutus TCP-pistokkeilla. Työssä rakennettiin testisovellus, jolla testattiin toteutuksen toimivuutta ja suorituskykyä.

Työn testisovellus koostui kahdesta osasta: WMS- ja PLC-ohjelmasta. WMS-ohjelma oli Java-ohjelmointikielellä kirjoitettu palvelin- ja asiakasohjelma, joka lähetti ja vastaanotti tavuvirtaviestejä TCP-pistokeyhteyden yli. PLC-ohjelma oli vastaava ohjelma toteutettuna Siemens S7-1500 -sarjan PLC:lle laitteen omalla ohjelmointikielellä.

Haastetta työssä aiheutti oma vähäinen kokemus PLC-ohjelmoinnista. Työssä esitellyt ohjelmat eivät ole loppuun asti hiottuja ja testattuja, vaan niiden toteutuksessa olisi vielä paljon parantamisen varaa.

Jos tehtäisiin uusi, vastaavanlainen sovellus kannattaisi rakennetta yksinkertaistaa niin, että WMS-ohjelmassa olisi vain yksi palvelinohjelma ja jokaisella PLC:llä olisi vain yksi asiakasohjelma. Palvelin odottaisi yhteydenottoja ja käynnistäisi jokaiselle PLC:lle oman säikeen kommunikointia varten.

Vaikka tässä työssä tämä toteutus todettiin sellaisenaan sopimattomaksi yleiseen laitekommunikointiin WMS-varastonhallintajärjestelmässä, sitä olisi kuitenkin tarkoitus kokeilla tulevassa projektissa pelkkään hälytysviestikommunikointiin PLC:n ja WMS:n välillä.

Koin itse oppineeni paljon PC:n ja PLC:n välisestä viestikommunikoinnista, kuten mitkä asiat onnistuvat helposti, ja missä erilaiset rajoitteet tekevät työn haastavaksi. Nämä opit tulevat varmasti pian tarpeeseen.

LÄHTEET

Anttila, A. 2001. TCP/IP-tekniikka. 2. korj. p. Helsinki: Helsinki Media.

Gosling, J. & McGilton, H. 1995. The Java Language Environment. [www-dokumentti]. Sun Microsystems. [Viitattu 29.4.2020]. Saatavana: http://www.stroustrup.com/1995_Java_whitepaper.pdf

Greanier, T. 2004. Java Foundations. San Francisco: Sybex.

Frystyk, H. 1994. The Internet Protocol Stack. [Verkkosivu]. W3 Consortium. [Viitattu 15.4.2020]. Saatavana: https://www.w3.org/People/Frystyk/thesis/TcpIp.html

JetBrains. Ei päiväystä. Features. [Verkkosivu]. JetBrains. [Viitattu 13.4.2020]. Saatavana: https://www.jetbrains.com/idea/features/

Pesmel. 2018. Pesmel 40 Years. [www-dokumentti]. Pesmel. [Viitattu 29.4.2020]. Saatavana: https://www.pesmel.com/sites/default/files/content_body_images/downloads/Pesmel_NewsFlow_Metal2018_40years.pdf

Postel, J. 1981. Internet Protocol, RFC 791. DARPA. California: Information Sciences Institute University of Southern California.

Reynders, D. & Wright, E. 2003. Practical TCP/IP and Ethernet Networking. Amsterdam: Newnes.

Siemens. 2020a. Simatic S7-1500. [Verkkosivu]. Siemens. [Viitattu 13.4.2020]. Saatavana: https://new.siemens.com/global/en/products/automation/systems/industrial/plc/simatic-s7-1500.html

Siemens. 2020b. TIA Portal -ohjelmointiympäristö. [Verkkosivu]. Siemens. [Viitattu 13.4.2020]. Saatavana: https://new.siemens.com/fi/fi/tuotteet/teollisuus/tia-portal.html

Siemens. 2020c. Basic Examples for Open User Communication. [Verkkosivu]. Siemens. [Viitattu 14.4.2020]. Saatavana: https://support.industry.siemens.com/cs/document/109747710/basic-examples-for-open-user-communication-(ouc)?dti=0&lc=en-GB

Siemens. 2020d. Examples of Open User Communication: TCP. [Verkkodokumentti]. Siemens. [Viitattu 14.4.2020]. Saatavana: https://support.industry.siemens.com/cs/attachments/109747710/109747710_Tcp_BaseComm_V1_en.pdf